CHAPTER 10

Getting and Managing Software

IN THIS CHAPTER

Installing software from the desktop

Working with RPM packaging

Using YUM to manage packages

Using rpm to work with packages

Installing software in the enterprise

In Linux distributions such as Fedora and Ubuntu, you don't need to know much about how software is packaged and managed to get the software you want. Those distributions have excellent software installation tools that automatically point to huge software repositories. Just a few clicks and you're using the software in little more time than it takes to download it.

The fact that Linux software management is so easy these days is a credit to the Linux community, which has worked diligently to create packaging formats, complex installation tools, and high-quality software packages. Not only is it easy to get the software, but after it's installed, it's easy to manage, query, update, and remove it.

This chapter begins by describing how to install software in Fedora using the new Software graphical installation tool. If you are just installing a few desktop applications on your own desktop system, you may not need much more than that and occasional security updates.

To dig deeper into managing Linux software, next I describe what makes up Linux software packages (comparing `deb` and `rpm` formatted packaging), underlying software management components, and commands (`dnf`, `yum`, and `rpm`) for managing software in Fedora and Red Hat Enterprise Linux. That's followed by a description of how to manage software packages in enterprise computing.

Managing Software on the Desktop

The Fedora Software window offers an intuitive way of choosing and installing desktop applications that does not align with typical Linux installation practices. The Ubuntu Software window offers the same interface for Ubuntu users. In either case, with the Software window, the smallest

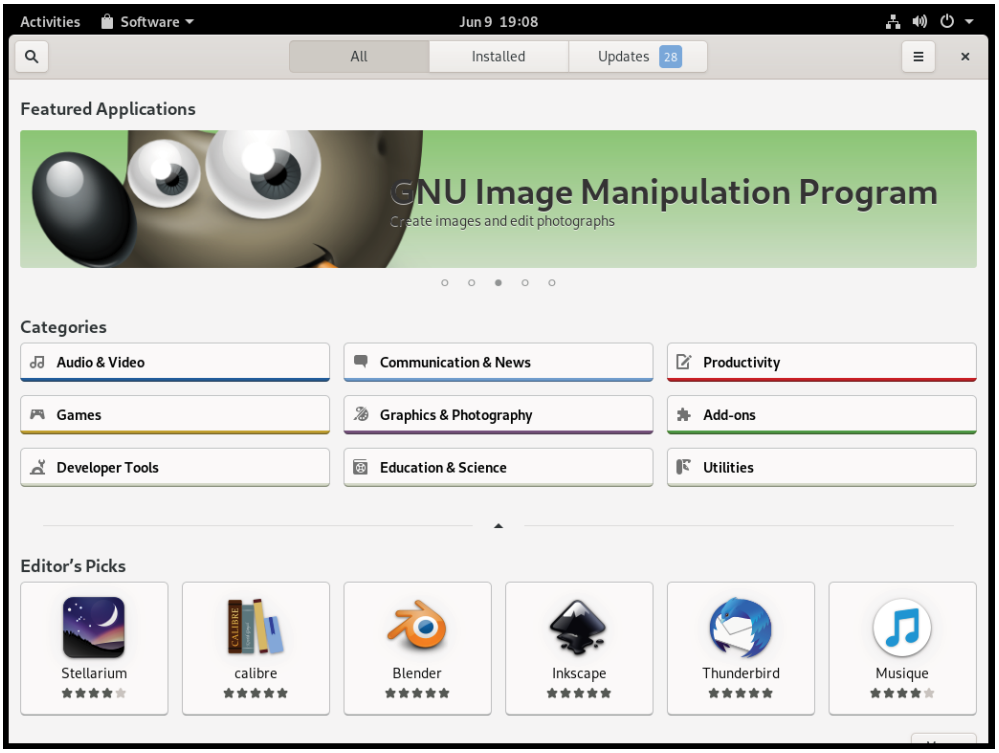
Linux® Bible, Tenth Edition. Christopher Negus.

software component you install is an application. With Linux, you install packages (such as `rpms` and `debs`).

Figure 10.1 shows an example of the Software window.

FIGURE 10.1

Install and manage software packages from the Software window.



To get to the Software window in either Fedora or Ubuntu, select Activities, then type **Software**, and press Enter. The first time you open this window, you can select Enable to allow third-party software repositories that are not part of the official redistributable Fedora repositories. Using the Software window is the best way to install desktop-oriented applications, such as word processors, games, graphics editors, and educational applications.

From the Software window, you can select the applications that you want to install from the Editor's Picks group (a handful of popular applications), choose from categories of applications (Audio & Video, Games, Graphics & Photography, and so on), or search by application

name or description. Select the Install button to have the Software window download and install all of the software packages needed to make the application work.

Other features of this window let you see all installed applications (Installed tab) or view a list of applications that have updated packages available for you to install (Updates tab). If you want to remove an installed application, simply click the Remove button next to the package name.

If you are using Linux purely as a desktop system, where you want to write documents, play music, and do other common desktop tasks, the Software window might be all you need to get the basic software you want. By default, your system connects to the main Fedora software repository and gives you access to hundreds of software applications. As noted earlier, you also have the option of accessing third-party applications that are still free for you to use but not redistribute.

Although the Software window lets you download and install hundreds of applications from the Fedora software repository, that repository actually contains tens of thousands of software packages. What packages can you not see from that repository, when might you want those other packages, and how can you gain access to those packages (as well as packages from other software repositories)?

Going Beyond the Software Window

If you are managing a single desktop system, you might be quite satisfied with the hundreds of packages that you can find through the Software window. Open-source versions of most common types of desktop applications are available to you through the Software window after you have a connection from Fedora to the Internet.

However, the following are some examples of why you might want to go beyond what you can do with the Software window:

More repositories Fedora and Red Hat Enterprise Linux distribute only open-source, freely distributable software. You may want to install some commercial software (such as Adobe Flash Player) or non-free software (available from repositories such as `rpmfusion.org`).

Beyond desktop applications Tens of thousands of software packages in the Fedora repository are not available through the Software window. Most of these packages are not associated with graphical applications at all. For example, some packages contain pure command-line tools, system services, programming tools, or documentation that doesn't show up in the Software window.

Flexibility Although you may not know it, when you install an application through the Software window, you may actually be installing multiple RPM packages. This set of packages may just be a default package set that includes documentation, extra fonts, additional software plug-ins, or multiple language packs that you

may or may not want. With `yum` and `rpm` commands, you have more flexibility on exactly which packages related to an application or other software feature is installed on your system.

More complex queries Using commands such as `yum` and `rpm`, you can get detailed information about packages, package groups, and repositories.

Software validation Using `rpm` and other tools, you can check whether a signed package has been modified before you installed it or whether any of the components of a package have been tampered with since the package was installed.

Managing software installation Although the Software window works well if you are installing desktop software on a single system, it doesn't scale well for managing software on multiple systems. Other tools are built on top of the `rpm` facility for doing that.

Before I launch into some of the command-line tools for installing and managing software in Linux, the next section describes how the underlying packaging and package management systems in Linux work. In particular, I focus on RPM packaging as it is used in Fedora, Red Hat Enterprise Linux, and related distributions as well as Deb packages, which are associated with Debian, Ubuntu, Linux Mint, and related distributions.

Understanding Linux RPM and DEB Software Packaging

On the first Linux systems, if you wanted to add software, you would grab the source code from a project that produced it, compile it into runnable binaries, and drop it onto your computer. If you were lucky, someone would have already compiled it in a form that would run on your computer.

The form of the package could be a tarball containing executable files (commands), documentation, configuration files, and libraries. (A *tarball* is a single file in which multiple files are gathered together for convenient storage or distribution.) When you install software from a tarball, the files from that tarball might be spread across your Linux system in appropriate directories (`/usr/share/man`, `/etc`, `/bin`, and `/lib`, to name just a few). Although it is easy to create a tarball and just drop a set of software onto your Linux system, this method of installing software makes it difficult to do these things:

Get dependent software You would need to know if the software you were installing depended on other software being installed for your software to work. Then you would have to track down that software and install that too (which might itself have some dependencies).

List the software Even if you knew the name of the command, you might not know where its documentation or configuration files were located when you looked for it later.

Remove the software Unless you kept the original tarball, or a list of files, you wouldn't know where all of the files were when it came time to remove them. Even if you knew, you would have to remove each one individually.

Update the software Tarballs are not designed to hold metadata about the contents that they contain. After the contents of a tarball are installed, you may not have a way to tell what version of the software you are using, making it difficult to track down bugs and get new versions of your software.

To deal with these problems, packages progressed from simple tarballs to more complex packaging. With only a few notable exceptions (such as Gentoo, Slackware, and a few others), the majority of Linux distributions went to one of two packaging formats—DEB and RPM:

DEB (.deb) packaging The Debian GNU/Linux project created `.deb` packaging, which is used by Debian and other distributions based on Debian (Ubuntu, Linux Mint, KNOPPIX, and so on). Using tools such as `apt-get`, `apt`, and `dpkg`, Linux distributions could install, manage, upgrade, and remove software.

RPM (.rpm) packaging Originally named Red Hat Package Manager, but later recursively renamed RPM Package Manager, RPM is the preferred package format for SUSE, Red Hat distributions (RHEL and Fedora), and those based on Red Hat distributions (CentOS, Oracle Linux, and so on). The `rpm` command was the first tool to manage RPMs. Later, `yum` was added to enhance the RPM facility, and now `dnf` is poised to eventually replace `yum`.

For managing software on individual systems, there are proponents on both sides of the RPM vs. DEB debate with valid points. Although RPM is the preferred format for managing enterprise-quality software installation, updates, and maintenance, DEB is very popular among many Linux enthusiasts. This chapter covers both RPM (Fedora and Red Hat Enterprise Linux) and (to some extent) DEB packaging and software management.

Understanding DEB packaging

Debian software packages hold multiple files and metadata related to some set of software in the format of an `ar` archive file. The files can be executables (commands), configuration files, documentation, and other software items. The metadata includes such things as dependencies, licensing, package sizes, descriptions, and other information. Multiple command-line and graphical tools are available for working with DEB files in Ubuntu, Debian, and other Linux distributions. Some of these include the following:

Ubuntu Software Center Select the Ubuntu Software application from the GNOME Activities menu. The window that appears lets you search for applications and packages that you want by searching for keywords or navigating categories.

aptitude The `aptitude` command is a package installation tool that provides a screen-oriented menu that runs in the shell. After you run the command, use arrow keys to highlight the selection you want, and press Enter to select it. You can upgrade packages, get new packages, or view installed packages.

apt* There is a set of `apt*` commands (`apt-get`, `apt`, `apt-config`, `apt-cache`, and so on) that can be used to manage package installation.

The Ubuntu Software Center is fairly intuitive for finding and installing packages. However, here are a few examples of commands that can help you install and manage packages with `apt*` commands. In this case, I'm looking for and installing the `vsftpd` package:

NOTE

Notice that the `apt*` commands are preceded by the `sudo` command in these examples. That's because it is common practice for an Ubuntu administrator to run administrative commands as a regular user with `sudo` privilege.

```
$ sudo apt-get update           Get the latest package versions
$ sudo apt-cache search vsftpd  Find package by key word (such as vsftpd)
$ sudo apt-cache show vsftpd    Display information about a package
$ sudo apt-get install vsftpd   Install the vsftpd package
$ sudo apt-get upgrade          Update installed packages if upgrade ready
$ sudo apt-cache pkgnames       List all packages that are installed
```

There are many other uses of `apt*` commands that you can try out. If you have an Ubuntu system installed, I recommend that you run `man apt` to get an understanding of what the `apt` and related commands can do.

Understanding RPM packaging

An *RPM package* is a consolidation of files needed to provide a feature, such as a word processor, a photo viewer, or a file server. The commands, configuration files, and documentation that make up the software feature can be inside an RPM. However, an RPM file also contains metadata that stores information about the contents of that package, where the package came from, what it needs to run, and other information.

What is in an RPM?

Before you even look inside an RPM, you can tell much about it by the name of the RPM package itself. To find out the name of an RPM package currently installed on your system (in this example the Firefox web browser), you could type the following from the shell in Fedora or Red Hat Enterprise Linux:

```
# rpm -q firefox
firefox-67.0-4.fc30.x86_64
```

From this, you can tell that the base name of the package is `firefox`. The version number is 67.0 (assigned by the upstream producer of Firefox, the Mozilla Project). The release

(assigned by the packager, Fedora, each time the package is rebuilt at the same release number) is 4. The `firefox` package was built for Fedora 30 (fc30) and is compiled for the x86 64-bit architecture (x86_64).

When the `firefox` package was installed, it was probably copied from the installation medium (such as a CD or DVD) or downloaded from a YUM repository (more on that later). If you had been given the RPM file and it was sitting in a local directory, the name would appear as `firefox-67.0-4.fc30.x86_64.rpm` and you could install it from there. Regardless of where it came from, once the package is installed, the name and other information about it are stored in an RPM database on the local machine.

To find out more about what is inside an RPM package, you can use options other than the `rpm` command to query that local RPM database, as in this example:

```
# rpm -qi firefox
Name        : firefox
Version     : 67.0
Release     : 4.fc30
Architecture: x86_64
Install Date: Sun 02 Jun 2019 09:37:25 PM EDT
Group       : Unspecified
Size        : 266449296
License     : MPLv1.1 or GPLv2+ or LGPLv2+
Signature   : RSA/SHA256, Fri 24 May 2019 12:09:57 PM EDT, Key ID
ef3c111fcfc659b9
Source RPM  : firefox-67.0-4.fc30.src.rpm
Build Date  : Thu 23 May 2019 10:03:55 AM EDT
Build Host  : buildhw-08.phx2.fedoraproject.org
Relocations : (not relocatable)
Packager    : Fedora Project
Vendor      : Fedora Project
URL         : https://www.mozilla.org/firefox/
Bug URL     : https://bugz.fedoraproject.org/firefox
Summary     : Mozilla Firefox Web browser
Description :
Mozilla Firefox is an open-source web browser, designed for standards
compliance, performance and portability.
```

Besides the information you got from the package name itself, the `-qi` (query information) option lets you see who built the package (Fedora Project), when it was built, and when it was installed. The group the package is in (Unspecified), its size, and the licensing are listed. To enable you to find out more about the package, the URL points to the project page on the Internet and the Summary and Description lines tell you what the package is used for.

Where do RPMs come from?

The software included with Linux distributions, or built to work with those distributions, comes from thousands of open-source projects all over the world. These projects, referred to

as *upstream software providers*, usually make the software available to anyone who wants it, under certain licensing conditions.

A Linux distribution takes the source code and builds it into binaries. Then it gathers those binaries together with documentation, configuration files, scripts, and other components available from the upstream provider.

After all of those components are gathered into the RPM, the RPM package is signed (so that users can test the package for validity) and placed in a repository of RPMs for the specific distribution and architecture (32-bit x86, 64-bit x86, and so on). The repository is placed on an installation CD or DVD or in a directory that is made available as an FTP, web, or NFS server.

Installing RPMs

When you initially install a Fedora or Red Hat Enterprise Linux system, many individual RPM packages make up that installation. After Linux is installed, you can add more packages using the Software window (as described earlier). Refer to Chapter 9, "Installing Linux," for information on installing Linux.

The first tool to be developed for installing RPM packages, however, was the `rpm` command. Using `rpm`, you can install, update, query, validate, and remove RPM packages. The command, however, has some major drawbacks:

Dependencies For most RPM packages to work, some other software (library, executables, and so on) must be installed on the system. When you try to install a package with `rpm`, if a dependent package is not installed, the package installation fails, telling you which components were needed. At that point, you have to dig around to find what package contained that component. When you go to install it, that dependent package might itself have dependencies that you need to install to get it to work. This situation is lovingly referred to as "dependency hell" and is often used as an example of why DEB packages were better than RPMs. DEB packaging tools were made to resolve package dependencies automatically, well before RPM-related packaging tools could do that.

Location of RPMs The `rpm` command expects you to provide the exact location of the RPM file when you try to install it. In other words, you would have to give `firefox-67.0-4.fc30.x86_64.rpm` as an option if the RPM were in the current directory or `http://example.com/firefox-67.0-4.fc30.x86_64.rpm` if it were on a server.

As Red Hat Linux and other RPM-based applications grew in popularity, it became apparent that something had to be done to make package installation more convenient. The answer was the YUM facility.

Managing RPM Packages with YUM

The *YellowDog Updater Modified (YUM)* project set out to solve the headache of managing dependencies with RPM packages. Its major contribution was to stop thinking about RPM packages as individual components and think of them as parts of larger software repositories.

With repositories, the problem of dealing with dependencies fell not to the person who installed the software but to the Linux distribution or third-party software distributor that makes the software available. So, for example, it would be up to the Fedora Project to make sure that every component needed by every package in its Linux distribution could be resolved by some other package in the repository.

Repositories could also build on each other. So, for example, the `rpmfusion.org` repository could assume that a user already had access to the main Fedora repository. If a package being installed from `rpmfusion.org` needed a library or command from the main Fedora repository, the Fedora package could be downloaded and installed at the same time that you install the `rpmfusion.org` package.

The yum repositories could be put in a directory on a web server (`http://`), on an FTP server (`ftp://`), on local media such as a CD or DVD, or in a local directory (`file://`). The locations of these repositories would then be stored on the user's system in the `/etc/yum.conf` file or, more typically, in separate configuration files in the `/etc/yum.repos.d` directory.

Transitioning from yum to dnf

The `dnf` command-line interface represents the next generation of YUM. DNF, which refers to itself as *Dandified YUM*, (`https://github.com/rpm-software-management/dnf/`) has been part of Fedora since version 18 and has just been added as the default RPM package manager for RHEL 8. Like `yum`, `dnf` is a tool for finding, installing, querying, and generally managing RPM packages from remote software repositories to your local Linux system.

While `dnf` maintains a basic command-line compatibility with `yum`, one of its main differences is that it adheres to a strict API. That API encourages the development of extensions and plug-ins to `dnf`.

For our purposes, almost all of the `yum` commands described in this chapter can be replaced by `dnf` or used as they are. The `yum` command is a symbolic link to `dnf` in Fedora and RHEL, so typing either command has the same result. For more information on DNF, refer to the DNF page at `https://dnf.readthedocs.io/`.

Understanding how yum works

This is the basic syntax of the `yum` command:

```
# yum [options] command
```

Using that syntax, you can find packages, see package information, find out about package groups, update packages, or delete packages, to name a few features. With the YUM repository and configuration in place, a user can install a package by simply typing something like the following:

```
# yum install firefox
```

The user only needs to know the package name (which could be queried in different ways, as described in the section "Searching for packages" later in this chapter). The YUM facility finds the latest version of that package available from the repository, downloads it to the local system, and installs it.

To gain more experience with the YUM facility, and to see where there are opportunities for you to customize how YUM works on your system, follow the descriptions of each phase of the YUM install process described next.

NOTE

In the latest Fedora and RHEL systems, the YUM configuration files are now actually links to DNF files in the `/etc/dnf` directory. Besides the main `dnf` configuration file (`/etc/dnf/dnf.conf`), that directory primarily contains modules and plug-ins that add to your ability to manage RPM packages.

1. **Checking `/etc/yum.conf`:**

   When any `yum` command starts, it checks the file `/etc/yum.conf` for default settings. The `/etc/yum.conf` file is the basic YUM configuration file. You can also identify the location of repositories here, although the `/etc/yum.repos.d` directory is the more typical location for identifying repositories. Here's an example of `/etc/yum.conf` on a RHEL 8 system, with a few others added:

   ```
   [main]
   gpgcheck=1
   installonly_limit=3
   clean_requirements_on_remove=True
   best=True

   cachedir=/var/cache/yum/$basearch/$releasever
   keepcache=0
   debuglevel=2
   logfile=/var/log/yum.log
   exactarch=1
   plugins=1
   ```

   The `gpgcheck` is used to validate each package against a key that you receive from those who built the RPM. It is on by default (`gpgcheck=1`). For packages in Fedora or RHEL, the key comes with the distribution to check all packages. To install packages that are not from your distribution, you need either to import the key to verify those packages or to turn off that feature (`gpgcheck=0`).

The `install_onlylimit=3` setting allows up to three versions of the same package to be kept on the system (don't set this to less than `2`, to ensure that you always have at least two kernel packages). The `clean_requirements_on_remove=True` tells `yum` to remove dependent packages when removing a package, if those packages are not otherwise required. With `best=True`, when upgrading a package, always try to use the highest version available.

Other settings that you can add to `yum.conf` tell YUM where to keep cache files (`/var/cache/yum`) and log entries (`/var/log/yum.log`) and whether to keep cache files around after a package is installed (`0` means no). You can raise the `debuglevel` value in the `yum.conf` file to above `2` if you want to see more details in your log files.

Next, you can see whether the exact architecture (x86, x86_64, and so on) should be matched when choosing packages to install (`1` means yes) and whether to use plug-ins (`1` means yes) to allow for things such as blacklists, white lists, or connecting to Red Hat Network for packages.

To find other features that you can set in the `yum.conf` file, type **`man yum.conf`**.

2. **Checking `/etc/yum.repos.d/*.repo` files:**

   Software repositories can be enabled by dropping files ending in `.repo` into the `/etc/yum.repos.d/` directory that point to the location of one or more repositories. In Fedora, even your basic Fedora repositories are enabled from `.repo` files in this directory. Here's an example of a simple yum configuration file named `/etc/yum.repos.d/myrepo.repo`:

   ```
   [myrepo]
   name=My repository of software packages
   baseurl=http://myrepo.example.com/pub/myrepo
   enabled=1
   gpgcheck=1
   gpgkey=file:///etc/pki/rpm-gpg/MYOWNKEY
   ```

   Each repository entry begins with the name of the repository enclosed in square brackets. The `name` line contains a human-readable description of the repository. The `baseurl` line identifies the directory containing the RPM files, which can be an `httpd://`, `ftp://`, or `file://` entry.

   The `enabled` line indicates whether the entry is active. A `1` is active; `0` is inactive. If there is no `enabled` line, the entry is active. The last two lines in the preceding code indicate whether to check the signatures on packages in this repository. The `gpgkey` line shows the location of the key that is used to check the packages in this repository.

   You can have as many repositories enabled as you like. However, keep in mind that when you use `yum` commands, every repository is checked and metadata about all packages is downloaded to the local system running the `yum` command. So, to be more efficient, don't enable repositories that you don't need.

3. **Downloading RPM packages and metadata from a YUM repository**:

   After `yum` knows the locations of the repositories, metadata from the `repodata` directory of each repository is downloaded to the local system. In fact, it is the existence of a `repodata` directory in a directory of RPMs that indicates that it is a YUM repository.

   Metadata information is stored on the local system in the `/var/cache/yum` directory. Any further queries about packages, package groups, or other information from the repository are gathered from the cached metadata until a time-out period is reached.

   After the time-out period is reached, `yum` retrieves fresh metadata if the `yum` command is run. By default, the time-out is 6 hours for `yum` and 48 hours for `dnf` minutes. You can change that period by setting `metadata_expire` in the `/etc/yum.conf` file.

   Next, `yum` looks at the packages that you requested to install and checks if any dependent packages are needed by those packages. With the package list gathered, `yum` asks you if it is okay to download all of those packages. If you choose yes, the packages are downloaded to the cache directories and installed.

4. **Installing RPM packages to Linux file system:**

   After all of the necessary packages are downloaded to the cache directories, `yum` runs `rpm` commands to install each package. If a package contains preinstall scripts (which might create a special user account or make directories), those scripts are run. The contents of the packages (commands, config files, docs, and so on) are copied to the filesystem at locations specified in the RPM metadata. Then any post install scripts are run. (Post install scripts run additional commands needed to configure the system after each package is installed.)

5. **Storing YUM repository metadata to local RPM database**.

   The metadata contained in each RPM package that is installed is ultimately copied into the local RPM database. The RPM database is contained in files that are stored in the `/var/lib/rpm` directory. After information about installed packages is in the local RPM database, you can do all sorts of queries of that database. You can see what packages are installed, list components of those packages, and see scripts or change logs associated with each package. You can even validate installed packages against the RPM database to see if anyone has tampered with installed components.

   The `rpm` command (described in the section "Installing, Querying, and Verifying Software with the `rpm` Command" later in this chapter) is the best tool for querying the RPM database. You can run individual queries with `rpm` or use it in scripts to produce reports or run common queries over and over again.

Now that you understand the basic functioning of the `yum` command, your Fedora system should be automatically configured to connect to the main Fedora repository and the Fedora Updates repository. You can try some `yum` command lines to install packages right now. Or, you can enable other third-party YUM repositories to draw software from.

Using YUM with third-party software repositories

The Fedora and Red Hat Enterprise Linux software repositories have been screened to contain only software that meets criteria that make it open and redistributable. In some instances, however, you may want to go beyond those repositories. Before you do, you should understand that some third-party repositories have these limitations:

- They may have less stringent requirements for redistribution and freedom from patent constraints than the Fedora and RHEL repositories have.
- They may introduce some software conflicts.
- They may include software that is not open source and, although it may be free for personal use, may not be redistributable.
- They may slow down the process of installing all of your packages (because metadata is downloaded for every repository you have enabled).

For those reasons, I recommend that you either (1) don't enable any extra software repositories, or (2) enable only the RPM Fusion repository (`http://rpmfusion.org`) at first for Fedora and the EPEL repository (`http://fedoraproject.org/wiki/EPEL`) for Red Hat Enterprise Linux. RPM Fusion represents a fusion of several popular third-party Fedora repositories (Freshrpms, `Livna.org`, and Dribble). See the repository's FAQ for details (`http://rpmfusion.org/FAQ`). To enable the Free RPM Fusion repository in Fedora, do the following:

1. Open a Terminal window.
2. Type **`su`** and enter the root password when prompted.
3. Type the following command on one line with no space in between the slash and `rpmfusion` (if that doesn't work, go to the `fedora` directory and choose the RPM appropriate for your version of Fedora):

   ```
   # rpm -Uvh http://download1.rpmfusion.org/free/fedora/
   rpmfusion-free-release-stable.noarch.rpm
   ```

The RPM Fusion Nonfree repository contains such things as codecs needed to play many popular multimedia formats. To enable the Nonfree repository in Fedora, type the following (again, type the following two lines on a single line, with no space between the two):

```
# rpm -Uhv http://download1.rpmfusion.org/nonfree/fedora/
rpmfusion-nonfree-release-stable.noarch.rpm
```

Most of the other third-party repositories that might interest you contain software that is not open source. For example, if you want to install the Adobe Flash plug-in for Linux, download the YUM repository package from Adobe, and you can use the `yum` command to install the Flash plug-in and get updates later by running the `yum update` command when updates are available.

Managing software with the yum command

The `yum` command has dozens of subcommands that you can use to work with RPM packages on your system. The following sections provide some examples of useful `yum` command

lines to search for, install, query, and update packages associated with your YUM repositories. The section "Installing and removing packages" describes how to install and remove installed packages with the `yum` command.

NOTE

Metadata, describing the contents of YUM repositories, is downloaded from each of your enabled YUM repositories the first time you run a `yum` command. Metadata is downloaded again after the `metadata_expire` time is reached. The more YUM repositories that you enable and the larger they are, the longer this download can take. You can reduce this download time by increasing the expire time (in the `/etc/yum.conf` file) or by not enabling repositories you don't need.

Searching for packages

Using different searching subcommands, you can find packages based on key words, package contents, or other attributes.

Let's say that you want to try out a different text editor but you can't remember the name of the one you wanted. You could start by using the `search` subcommand to look for the term *editor* in the name or description:

```
# yum search editor
...
eclipse-veditor.noarch : Eclipse-based Verilog/VHDL plugin
ed.x86_64 : The GNU line editor
emacs.x86_64 : GNU Emacs text editor
```

The search uncovered a long list of packages containing *editor* in the name or description. The one I was looking for is named `emacs`. To get information about that package, I can use the `info` subcommand:

```
# yum info emacs
Name         : emacs
Epoch        : 1
Version      : 26.2
Release      : 1.fc30
Architecture : x86_64
Size         : 3.2 M
Source       : emacs-26.2-1.fc30.src.rpm
Repository   : updates
Summary      : GNU Emacs text editor
URL          : http://www.gnu.org/software/emacs/
License      : GPLv3+ and CC0-1.0
Description  : Emacs is a powerful, customizable, self-documenting,
modeless text
             : editor. Emacs contains special code editing features,
a scripting
```

```
              : language (elisp), and the capability to read mail,
news, and more
              : without leaving the editor.
```

If you know the command, configuration file, or library name you want but don't know what package it is in, use the `provides` subcommand to search for the package. Here you can see that the `dvdrecord` command is part of the `wodim` package:

```
# yum provides dvdrecord
wodim-1.1.11-41.fc30.x86_64 : A command line CD/DVD recording program
Repo        : fedora
Matched from:
Filename    : /usr/bin/dvdrecord
```

The `list` subcommand can be used to list package names in different ways. Use it with a package base name to find the version and repository for a package. You can list just packages that are `available` or `installed`, or you can list `all` packages.

```
# yum list emacs
emacs.i686     1:26.2-1.fc30     updates
# yum list available
CUnit.i686     2.1.3-17.el8   rhel-8-for-x86_64-appstream-rpms
CUnit.x86_64     2.1.3-17.el8   rhel-8-for-x86_64-appstream-rpms
GConf2.i686      3.2.6-22.el8   rhel-8-for-x86_64-appstream-rpms
LibRaw.i686      0.19.1-1.el8   rhel-8-for-x86_64-appstream-rpm
...
# yum list installed
Installed Packages
GConf2.x86_64       3.2.6-22.el8      @AppStream
ModemManager.x86_64 1.8.0-1.el8       @anaconda
...
# yum list all
...
```

If you find a package but you want to see what components that package is dependent on, you can use the `deplist` subcommand. With `deplist`, you can see the components (dependency) but also the package that component comes in (provider). Using `deplist` can help if no package is available to provide a dependency, but you want to know what the component is so that you can search other repositories for it. Consider the following example:

```
# yum deplist emacs | less
package: emacs-1:26.1-8.fc30.x86_64
  dependency: /bin/sh
   provider: bash-5.0.7-1.fc30.i686
   provider: bash-5.0.7-1.fc30.x86_64
  dependency: /usr/sbin/alternatives
   provider: alternatives-1.11-4.fc30.x86_64
```

Installing and removing packages

The `install` subcommand lets you install one or more packages, along with any dependent packages needed. With `yum install`, multiple repositories can be searched to fulfill needed dependencies. Consider the following example of `yum install`:

```
# yum install emacs
...
Package                 Architecture Version            Repository  Size
================================================================================
Installing:
 emacs                   x86_64       1:26.2-1.fc30      updates    3.2 M
Installing dependencies:
 emacs-common            x86_64       1:26.2-1.fc30      updates     38 M
 ImageMagick-libs        x86_64       1:6.9.10.28-1.fc30 fedora     2.2 M
 fftw-libs-double        x86_64       3.3.8-4.fc30       fedora     984 k
 ...

Transaction Summary
================================================================================
Install  7 Packages

Total download size: 45 M
Installed size: 142 M
Is this ok [y/N]: y
```

You can see here that `emacs` requires that `emacs-common` and several other packages be installed so all are queued up for installation. The six packages together are 45MB to download, but they consume 142MB after installation. Pressing **y** installs them. You can put a `-y` on the command line (just after the `yum` command) to avoid having to press **y** to install the packages. Personally, however, I usually want to see all of the packages about to be installed before I agree to the installation.

You can reinstall a package if you mistakenly delete components of an installed package. If you attempt a regular install, the system responds with "nothing to do." You must instead use the `reinstall` subcommand. For example, suppose that you installed the `zsh` package and then deleted `/bin/zsh` by mistake. You could restore the missing components by typing the following:

```
# yum reinstall zsh
```

You can remove a single package, along with its dependencies that aren't required by other packages, with the `remove` subcommand. For example, to remove the `emacs` package and dependencies, you could type the following:

```
# yum remove emacs
Removing:
 emacs                   x86_64       1:26.2-1.fc30      updates     38 M
```

```
Removing unused dependencies:
 ImageMagick-libs        x86_64       1:6.9.10.28-1.fc30 fedora     8.9 M
 emacs-common            x86_64       1:26.2-1.fc30      updates     89 M
 fftw-libs-double        x86_64       3.3.8-4.fc30       fedora     4.2 M
 ...

Transaction Summary
================================================================================
Remove  7 Packages

Freed space: 142 M
Is this ok [y/N]: y
```

Notice that the space shown for each package is the actual space consumed by the package in the file system and not the download size (which is considerably smaller).

An alternative method to remove a set of packages that you have installed is to use the `history` subcommand. Using `history`, you can see your `yum` activities and undo an entire transaction. In other words, all of the packages that you installed can be uninstalled using the `undo` option of the `history` subcommand, as in the following example:

```
# yum history
ID     | Command line   | Date and time    | Action(s)      | Altered
-------------------------------------------------------------------------------
    12 | install emacs  | 2019-06-22 11:14 | Install        |    7
...
# yum history info 12
Transaction ID : 12
...
Command Line    : install emacs
...
# yum history undo 12
Undoing transaction 12, from Sat 22 Jun 2019 11:14:42 AM EDT

Install emacs-1:26.2-1.fc30.x86_64                 @updates
    Install emacs-common-1:26.2-1.fc30.x86_64          @updates
 ...
```

Before you undo the transaction, you can view the transaction to see exactly which packages were involved. Viewing the transaction can save you from mistakenly deleting packages that you want to keep. By undoing transaction `12`, you can remove all packages that were installed during that transaction. If you are trying to undo an install that included dozens or even hundreds of packages, `undo` can be a very useful option.

Updating packages

As new releases of a package become available, they are sometimes put into separate update repositories or simply added to the original repository. If multiple versions of a package are available (whether in the same repository or in another enabled repository), `yum` provides the latest version when you install a package. For some packages, such as the Linux kernel package, you can keep multiple versions of the same package.

If a new version of a package shows up later, you can download and install the new version of the package by using the `update` subcommand.

The `check-update` subcommand can check for updates. The `update` subcommand can be used to update a single package or to get updates to all packages that are currently installed and have an update available. Or, you can simply update a single package (such as the `cups` package), as in this example:

```
# yum check-update
...
file.x86_64       5.36-3.fc30     updates
file-libs.x86_64  5.36-3.fc30     updates
firefox.x86_64    67.0.4-1.fc30   updates
firewalld.noarch  0.6.4-1.fc30    updates
...
# yum update
Dependencies resolved.
================================================================
 Package             Arch   Version         Repository    Size
================================================================
Upgrading:
 NetworkManager      x86_64 1:1.16.2-1.fc30 updates      1.7 M
 NetworkManager-adsl x86_64 1:1.16.2-1.fc30 updates       25 k
...
Transaction Summary
================================================================
Install     7 Packages
Upgrade   172 Package(s)
Total download size: 50 M
Is this ok [y/N]: y
# yum update cups
```

The preceding command requested to update the `cups` package. If other dependent packages need to be updated to update `cups`, those packages would be downloaded and installed as well.

Updating groups of packages

To make it easier to manage a whole set of packages at once, YUM supports package groups. For example, you could install GNOME Desktop Environment (to get a whole desktop) or Virtualization (to get packages needed to set up the computer as a virtualization host). You can start by running the `grouplist` subcommand to see a list of group names:

```
# yum grouplist | less
Available Environment Groups:
   Fedora Custom Operating System
   Minimal Install
   Fedora Server Edition
...
Installed Groups:
   LibreOffice
   GNOME Desktop Environment
   Fonts
...
Available Groups:
   Authoring and Publishing
   Books and Guides
   C Development Tools and Libraries
...
```

Let's say that you wanted to try out a different desktop environment. You see LXDE, and you want to know what is in that group. To find out, use the `groupinfo` subcommand:

```
# yum groupinfo LXDE
Group: LXDE
 Description: LXDE is a lightweight X11 desktop environment...
 Mandatory Packages:
...
   lxde-common
   lxdm
   lxinput
   lxlauncher
   lxmenu-data
...
```

In addition to showing a description of the group, `groupinfo` shows Mandatory Packages (those that are always installed with the group), Default Packages (those that are installed by default but can be excluded), and Optional Packages (which are part of the group but not installed by default). When you use some graphical tools to install package groups, you can uncheck default packages or check optional packages to change whether they are installed with the group.

If you decide that you want to install a package group, use the `groupinstall` subcommand:

```
# yum groupinstall LXDE
```

This groupinstall resulted in 101 packages from the group being installed and 5 existing packages being updated. If you decide that you don't like the group of packages, you can remove the entire group at once using the groupremove subcommand:

```
# yum groupremove LXDE
```

Maintaining your RPM package database and cache

Several subcommands to yum can help you do maintenance tasks, such as checking for problems with your RPM database or clearing out the cache. The YUM facility has tools for maintaining your RPM packages and keeping your system's software efficient and secure.

Clearing out the cache is something that you want to do from time to time. If you decide to keep downloaded packages after they are installed (they are removed by default, based on the keepcache=0 setting in the /etc/yum.conf file), your cache directories (under /var/cache/yum) can fill up. Metadata stored in cache directories can be cleared, causing fresh metadata to be downloaded from all enabled YUM repositories the next time yum is run. Here are ways to clear that information:

```
# yum clean packages
14 files removed
# yum clean metadata
Cache was expired
16 files removed
# yum clean all
68 files removed
```

Although unlikely, it's possible that the RPM database can become corrupted. This can happen if something unexpected occurs, such as pulling out the power cord when a package is partially installed. You can check the RPM database to look for errors (yum check) or just rebuild the RPM database files. Here's an example of a corrupt RPM database and the rpm command that you can use to fix it:

```
# yum check
error: db5 error(11) from dbenv->open: Resource temporarily
unavailable
error: cannot open Packages index using db5-Resource temporarily
unavailable(11)
error: cannot open Packages database in /var/lib/rpm
Error: Error: rpmdb open failed
# rpm --rebuilddb
# yum check
```

The yum clean examples in the preceding command lines remove cached data from the /var/cache/yum subdirectories. The rpm --rebuilddb example rebuilds the database. The yum check example can check for problems with the local RPM cache and database but notice that we used the rpm command to fix the problem.

In general, the command best suited for working with the local RPM database is the rpm command.

Downloading RPMs from a YUM repository

If you just want to examine a package without actually installing it, you can download that package with the `dnf` command or, in earlier releases, use the `yumdownloader` command. Either case causes the package that you name to be downloaded from the YUM repository and copied to your current directory.

For example, to download the latest version of the Firefox web browser package with `yumdownloader` from the YUM repository to your current directory, type the following:

```
# yumdownloader firefox
firefox-67.0.4-1.fc30.x86_64.rpm    6.1 MB/s |  92 MB     00:14
```

To use the `dnf` command, type this:

```
# dnf download firefox
firefox-60.7.0-1.el8_0.x86_64.rpm   6.1 MB/s |  93 MB     00:15
```

With any downloaded RPM package now sitting in your current directory, you can use a variety of `rpm` commands to query or use that package in different ways (as described in the next section).

Installing, Querying, and Verifying Software with the rpm Command

There is a wealth of information about installed packages in the local RPM database. The `rpm` command contains dozens of options to enable you to find information about each package, such as the files it contains, who created it, when it was installed, how large it is, and many other attributes. Because the database contains fingerprints (md5sums) of every file in every package, it can be queried with RPM to find out if files from any package have been tampered with.

The `rpm` command can still do basic install and upgrade activities, although most people only use `rpm` in that way when there is a package sitting in the local directory, ready to be installed. So, let's get one in our local directory to work with. Type the following to download the latest version of the `zsh` package:

```
# dnf download zsh
zsh-5.5.1-6.el8.x86_64.rpm     3.0 MB/s | 2.9 MB   00:00
```

With the `zsh` package downloaded to your current directory, try some `rpm` commands on it.

Installing and removing packages with rpm

To install a package with the `rpm` command, type the following:

```
# rpm -i zsh-5.5.1-6.el8.x86_64.rpm
```

Notice that the entire package name is given to install with `rpm`, not just the package base name. If an earlier version of `zsh` were installed, you could upgrade the package using `-U`. Often, people use `-h` and `-v` options to get hash signs printed and more verbose output during the upgrade:

```
# rpm -Uhv zsh-5.5.1-6.el8.x86_64.rpm
Verifying...          ########################## [100%]
Preparing...          ########################## [100%]
   1:zsh-5.5.1-6.el8  ########################## [100%]
```

Although an install (`-i`) only installs a package if the package is not already installed, an upgrade (`-U`) installs the package even if it is already installed. A third type of install, called freshen (`-F`), installs a package only if an existing, earlier version of a package is installed on the computer, as in this example:

```
# rpm -Fhv *.rpm
```

You could use the previous freshen command if you were in a directory containing thousands of RPMs but only wanted to update those that were already installed (in an earlier version) on your system and skip those that were not yet installed.

You can add a few interesting options to any of your install options. The `--replacepkgs` option enables you to reinstall an existing version of a package (if, for example, you had mistakenly deleted some components), and the `--oldpackage` enables you to replace a newer package with an earlier version.

```
# rpm -Uhv --replacepkgs emacs-26.1-5.el8.x86_64.rpm
# rpm -Uhv --oldpackage zsh-5.0.2-25.el7_3.1.x86_64.rpm
```

You can remove a package with the `-e` option. You only need the base name of a package to remove it. Here's an example:

```
# rpm -e emacs
```

The `rpm -e emacs` command would be successful because no other packages are dependent on `emacs`. However, it would leave behind `emacs-common`, which was installed as a dependency to `emacs`. If you had tried to remove `emacs-common` first, that command would fail with a "Failed dependencies" message.

Querying rpm information

After the package is installed, you can query for information about it. Using the `-q` option, you can see information about the package including a description `(-qi)`, list of files `(-ql)`, documentation `(-qd)`, and configuration files `(-qc)`.

```
# rpm -qi zsh
Name        : zsh
Version     : 5.5.1
Release     : 6.el8
...
```

```
# rpm -ql zsh
/etc/skel/.zshrc
/etc/zlogin
/etc/zlogout
...
# rpm -qd zsh
/usr/share/doc/zsh/BUGS
/usr/share/doc/zsh/CONTRIBUTORS
/usr/share/doc/zsh/FAQ
...
# rpm -qc zsh
/etc/skel/.zshrc
/etc/zlogin
/etc/zlogout
...
```

You can use options to query any piece of information contained in an RPM. You can find what an RPM needs for it to be installed (`--requires`), what version of software a package provides (`--provides`), what scripts are run before and after an RPM is installed or removed (`--scripts`), and what changes have been made to an RPM (`--changelog`).

```
# rpm -q --requires emacs-common
/bin/sh
/usr/bin/pkg-config
/usr/sbin/alternatives
...
# rpm -q --provides emacs-common
config(emacs-common) = 1:26.2-1.fc30
emacs-common = 1:26.2-1.fc30
emacs-common(x86-64) = 1:26.2-1.fc30
emacs-el = 1:26.2-1.fc30
pkgconfig(emacs) = 1:26.2
# rpm -q --scripts httpd
postinstall scriptlet (using /bin/sh):
if [ $1 -eq 1 ] ; then
        # Initial installation
        systemctl --no-reload preset httpd.service...
...
# rpm -q --changelog httpd | less
* Thu May 02 2019 Lubos Uhliarik <luhliari@redhat.com> - 2.4.39-4
- httpd dependency on initscripts is unspecified (#1705188)
* Tue Apr 09 2019 Joe Orton <jorton@redhat.com> - 2.4.39-3
...
```

In the previous two examples, you can see that scripts inside the `httpd` package uses `systemctl` command to set up the `httpd` service. The `--changelog` option enables you to see why changes have been made to each version of the package.

Using a feature called `--queryformat`, you can query different tags of information and output them in any form you like. Run the `--querytags` option to be able to see all of the tags that are available:

```
# rpm --querytags | less
ARCH
ARCHIVESIZE
BASENAMES
BUGURL
...
# rpm -q binutils --queryformat "The package is %{NAME} \
     and the release is %{RELEASE}\n"
The package is binutils and the release is 29.fc30
```

All of the queries that you have done so far have been to the local RPM database. By adding a `-p` to those query options, you can query an RPM file sitting in your local directory instead. The `-p` option is a great way to look inside a package that someone gives you to investigate what it is before you install it on your system.

If you haven't already, get the `zsh` package and put it in your local directory (`dnf download zsh`). Then run some query commands on the RPM file.

```
# rpm -qip zsh-5.7.1-1.fc30.x86_64.rpm View info about the RPM file
# rpm -qlp zsh-5.7.1-1.fc30.x86_64.rpm List all files in the RPM file
# rpm -qdp zsh-5.7.1-1.fc30.x86_64.rpm Show docs in the RPM file
# rpm -qcp zsh-5.7.1-1.fc30.x86_64.rpm List config files in the RPM file
```

Verifying RPM packages

Using the `-V` option, you can check the packages installed on your system to see if the components have been changed since the packages were first installed. Although it is normal for configuration files to change over time, it is not normal for binaries (the commands in `/bin`, `/sbin`, and so on) to change after installation. Binaries that are changed are probably an indication that your system has been cracked.

In this example, I'm going to install the `zsh` package and mess it up. If you want to try along with the examples, be sure to remove or reinstall the package when you are finished.

```
# rpm -i zsh-5.7.1-1.fc30.x86_64.rpm
# echo hello > /bin/zsh
# rm /etc/zshrc
# rpm -V zsh
missing   c  /etc/zshrc
S.5....T.    /bin/zsh
```

In this output, you can see that the `/bin/zsh` file has been tampered with and `/etc/zshrc` has been removed. Each time that you see a letter or a number instead of a dot from

the `rpm -V` output, it is an indication of what has changed. Letters that can replace the dots (in order) include the following:

```
S    file Size differs
M    Mode differs (includes permissions and file type)
5    MD5 sum differs
D    Device major/minor number mismatch
L    readLink(2) path mismatch
U    User ownership differs
G    Group ownership differs
T    mTime differs
P    caPabilities differ
```

Those indicators are from the Verify section of the `rpm` man page. In my example, you can see the file size has changed (`S`), the md5sum checked against the file's fingerprint has changed (`5`), and the modification time (`T`) on the file differs.

To restore the package to its original state, use `rpm` with the `--replacepkgs` option, as shown next. (The `yum reinstall zsh` command would work as well.) Then check it with `-V` again. No output from `-V` means that every file is back to its original state.

```
# rpm -i --replacepkgs zsh-5.7.1-1.fc30.x86_64.rpm
# rpm -V zsh
```

It is good practice to back up your RPM database (from `/var/lib/rpm`) and copy it to some read-only medium (such as a CD). Then, when you go to verify packages that you suspect were cracked, you know that you aren't checking it against a database that has also been cracked.

Managing Software in the Enterprise

At this point, you should have a good working knowledge of how to install, query, remove, and otherwise manipulate packages with graphical tools, the `yum` command, and the `rpm` command. When you start working with RPM files in a large enterprise, you need to extend that knowledge.

Features used to manage RPM packages in the enterprise with Red Hat Enterprise Linux offer a bit more complexity and much more power. Instead of having one big software repository, as Fedora does, RHEL provides software using Red Hat Subscription Management, requiring paid subscriptions/entitlements that allow access to a variety of software channels and products (RHEL, Red Hat Virtualization, Red Hat Software Collections, and so on).

In terms of enterprise computing, one of the great benefits of the design of RPM packages is that their management can be automated. Other Linux packaging schemes allow packages to stop and prompt you for information when they are being installed (such as asking for a

directory location or a username). RPM packages install without interruption, offering some of the following advantages:

Kickstart files All of the questions that you answer during a manual install, and all of the packages that you select, can be added into a file called a *kickstart file*. When you start a Fedora or Red Hat Enterprise Linux installer, you can provide a kickstart file at the boot prompt. From that point on, the entire installation process completes on its own. Any modifications to the default package installs can be made by running pre and post scripts from the kickstart file to do such things as add user accounts or modify configuration files.

PXE boot You can configure a PXE server to allow client computers to boot an anaconda (installer) kernel and a select kickstart file. A completely blank computer with a network interface card (NIC) that supports PXE booting can simply boot from its NIC to launch a fresh installation. In other words, turn on the computer, and if it hits the NIC in its boot order, a few minutes later you can have a freshly installed system, configured to your exact specifications without intervention.

Satellite server (Spacewalk) Red Hat Enterprise Linux systems can be deployed using what is referred to as *Satellite Server*. Built into Satellite Server are the same features that you have from Red Hat CDN to manage and deploy new systems and updates. RHEL systems can be configured to get automatic software updates at times set from the satellite server. Sets of packages called Errata that fix specific problems can be quickly and automatically deployed to the systems that need them.

Container Images Instead of installing individual RPMs on a system, you can package up a few or a few hundred RPMs into a container image. The container image is like an RPM in that it holds a set of software, but unlike an RPM in that it is more easily added to a system, run directly, and deleted from a system than an RPM is.

Descriptions of how to use kickstart files, Satellite Servers, containers and other enterprise-ready installation features are beyond the scope of this book. But the understanding you have gained from learning about YUM and RPM remain critical components of all of the features just mentioned.

Summary

Software packaging in Fedora, Red Hat Enterprise Linux, and related systems is provided using software packages based on the RPM Package Manager (RPM) tools. Debian, Ubuntu, and related systems package software into DEB files. You can try easy-to-use graphical tools such as the Software window for finding and installing packages. The primary command-line tools include the `yum`, `dnf`, and `rpm` commands for Red Hat–related systems and `aptitude`, `apt*` and `dpkg` for Debian-related systems.

Using these software management tools, you can install, query, verify, update, and remove packages. You can also do maintenance tasks, such as clean out cache files and rebuild the

RPM database. This chapter describes many of the features of the Software window, as well as `yum`, `dnf`, and `rpm` commands.

With your system installed and the software packages that you need added, it's time to configure your Fedora, RHEL, Debian, or Ubuntu system further. If you expect to have multiple people using your system, your next task could be to add and otherwise manage user accounts on your system. Chapter 11, "Managing User Accounts," describes user management in Fedora, RHEL, and other Linux systems.

Exercises

These exercises test your knowledge of working with RPM software packages in Fedora or Red Hat Enterprise Linux. To do the exercises, I recommend that you have a Fedora system in front of you that has an Internet connection. (Most of the procedures work equally well on a registered RHEL system.)

You need to be able to reach the Fedora repositories (which should be set up automatically). If you are stuck, solutions to the tasks are shown in Appendix B (although in Linux, there are often multiple ways to complete a task).

1. Search the YUM repository for the package that provides the `mogrify` command.
2. Display information about the package that provides the `mogrify` command, and determine what is that package's home page (URL).
3. Install the package containing the `mogrify` command.
4. List all of the documentation files contained in the package that provides the `mogrify` command.
5. Look through the `changelog` of the package that provides the `mogrify` command.
6. Delete the `mogrify` command from your system and verify its package against the RPM database to see that the command is indeed missing.
7. Reinstall the package that provides the `mogrify` command, and make sure that the entire package is intact again.
8. Download the package that provides the `mogrify` command to your current directory.
9. Display general information about the package you just downloaded by querying the package's RPM file in the current directory.
10. Remove the package containing the `mogrify` command from your system.